Outline

1. From Optimal Interpolation to 3D-Var
2. The Maximum Likelihood Approach
3. Minimisation
4. Summary
Outline

1. From Optimal Interpolation to 3D-Var
2. The Maximum Likelihood Approach
3. Minimisation
4. Summary
From Optimal Interpolation to 3D-Var

- Previously in “Assimilation Algorithms”: linear analysis equation

\[ x_a = x_b + K [ y - \mathcal{H}(x_b) ] \]

where

\[ K = P^b H^T \left[ H P^b H^T + R \right]^{-1} \equiv \left[ P^{b^{-1}} + H^T R^{-1} H \right]^{-1} H^T R^{-1} \]

- Optimal Interpolation (OI) applies direct solution methods to invert the matrix \( HP^b H^T + R \).
- Data selection is applied to reduce the dimension of the matrix.
- Direct methods require access to the matrix elements. In particular, \( HP^b H^T \) must be available in matrix form.
- This limits us to very simple observation operators.
From Optimal Interpolation to 3D-Var

- Linear analysis equation: \( \mathbf{x}_a = \mathbf{x}_b + \mathbf{K} [ \mathbf{y} - \mathcal{H}(\mathbf{x}_b) ] \)
- For \( \mathbf{K} = \mathbf{P}^b \mathbf{H}^\mathsf{T} \left[ \mathbf{H} \mathbf{P}^b \mathbf{H}^\mathsf{T} + \mathbf{R} \right]^{-1} \)

we have

\[
\mathbf{x}_a = \mathbf{x}_b + \mathbf{P}^b \mathbf{H}^\mathsf{T} \left[ \mathbf{H} \mathbf{P}^b \mathbf{H}^\mathsf{T} + \mathbf{R} \right]^{-1} \left[ \mathbf{y} - \mathcal{H}(\mathbf{x}_b) \right]
\]

if

\[
\mathbf{z} = \left[ \mathbf{H} \mathbf{P}^b \mathbf{H}^\mathsf{T} + \mathbf{R} \right]^{-1} \left[ \mathbf{y} - \mathcal{H}(\mathbf{x}_b) \right]
\]

then

\[
\mathbf{x}_a = \mathbf{x}_b + \mathbf{P}^b \mathbf{H}^\mathsf{T} \mathbf{z}
\]

and we have to solve

\[
\left[ \mathbf{H} \mathbf{P}^b \mathbf{H}^\mathsf{T} + \mathbf{R} \right] \mathbf{z} = \mathbf{y} - \mathcal{H}(\mathbf{x}_b)
\]
From Optimal Interpolation to 3D-Var

- Linear analysis equation: $x_a = x_b + K[y - \mathcal{H}(x_b)]$
- For $K = \left( P^{-1}b + HT^{-1}R \right)^{-1}HT^{-1}$

we have $x_a = x_b + \left( P^{-1}b + HT^{-1}R \right)^{-1}HT^{-1} \left[ y - \mathcal{H}(x_b) \right]$

if $\delta x = \left( P^{-1}b + HT^{-1}R \right)^{-1}HT^{-1} \left[ y - \mathcal{H}(x_b) \right]$

then $x_a = x_b + \delta x$

and we have to solve $\left( P^{-1}b + HT^{-1}R \right) \delta x = HT^{-1} \left[ y - \mathcal{H}(x_b) \right]$
From Optimal Interpolation to 3D-Var

There are two forms to solve the linear analysis equation, depending which expression we adopt for $K$:

- For $K = P^b H^T \left[ H P^b H^T + R \right]^{-1}$ we have $x_a = x_b + P^b H^T z$ and:

\[
\begin{bmatrix}
HP^b H^T + R
\end{bmatrix} z = y - \mathcal{H}(x_b)
\]

- For $K = \left[ P^b^{-1} + H^T R^{-1} H \right]^{-1} H^T R^{-1}$, we have $x_a = x_b + \delta x$ and:

\[
\begin{bmatrix}
P^b^{-1} + H^T R^{-1} H
\end{bmatrix} \delta x = H^T R^{-1} \left[ y - \mathcal{H}(x_b) \right]
\]

The linear analysis equation could be solved as an equation of the form:

\[
A x = b
\]

The first of these alternatives is called PSAS

The second (although it may not look like it yet) is 3D-Var
From Optimal Interpolation to 3D-Var

<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>✗ Find the solution $x_a$ of the linear system:</td>
</tr>
<tr>
<td>$Ax = b$.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Direct methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>✗ Direct methods attempt to solve the problem by a finite sequence of operations.</td>
</tr>
<tr>
<td>✗ In the absence of rounding errors, direct methods would deliver an exact solution $x_a$ of the linear system.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iterative methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>✗ Beginning with an approximation to the solution $x_0$, an iterative method is a mathematical procedure that generates a sequence of improving approximate solutions $x_1, x_2, \ldots x_n$.</td>
</tr>
<tr>
<td>✗ The $n$-th approximation is derived from the previous ones.</td>
</tr>
<tr>
<td>✗ The sequence of solutions converges to the exact solution.</td>
</tr>
</tbody>
</table>
From Optimal Interpolation to 3D-Var

- Iterative methods have significant advantages over the direct methods used in OI.
- They can be applied to much larger problems than direct techniques, so we can avoid data selection.
- They do not require access to the matrix elements.
- Typically, to solve $\mathbf{Ax} = \mathbf{b}$, requires only the ability to calculate matrix-vector products: $\mathbf{Ax}$.
- This allows us to use operators defined by pieces of code rather than explicitly as matrices.
- Examples of such operators include radiative-transfer codes, numerical models, Fourier transforms, etc.
Example: Conjugate Gradients

To solve $A\mathbf{x} = \mathbf{b}$, where $A$ is real, symmetric and positive-definite:

$r_0 := \mathbf{b} - A\mathbf{x}_0 \quad p_0 := r_0 \quad k := 0$

while $r_{k+1}$ is too large do

/* Step in the direction of $p_k$ */
$\alpha_k := \frac{r_k^T r_k}{p_k^T A p_k}$;

/* New state */
$x_{k+1} := x_k + \alpha_k p_k$;

/* New residual */
$r_{k+1} := r_k - \alpha_k A p_k$;

/* New direction of descent */
$\beta_k := \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$;

$p_{k+1} := r_{k+1} + \beta_k p_k$;

/* Next iteration */
$k := k + 1$;

end

The result is $x_{k+1}$
Outline

1. From Optimal Interpolation to 3D-Var

2. The Maximum Likelihood Approach

3. Minimisation

4. Summary
As we have seen, (linear) 3D-Var analysis can be seen as an application of iterative solution methods to the linear analysis equation.

Historically, 3D-Var was not developed this way.

We will now consider this alternative derivation.

We will need to apply Bayes’ theorem:

\[
p(A|B) = \frac{p(B|A)p(A)}{p(B)}
\]

where \(p(A|B)\) is the probability of \(A\) given \(B\), etc.
Maximum Likelihood

- We developed the linear analysis equation by searching for a linear combination of observation and background that minimised the variance of the error.

- An alternative approach is to look for the most probable solution \( x_a \), given the observations \( Y \) and having a prior knowledge \( x_b \) on the solution:

\[
x_a = \arg \max_x \left[ p(x|y) \right]
\]

- It will be convenient to define a cost function

\[
J(x) = -\log \left[ p(x|y) \right] + \text{const}.
\]

- Then, since log is a monotonic function:

\[
x_a = \arg \min_x \left[ J(x) \right]
\]
Maximum Likelihood

- Applying Bayes’ theorem gives:

\[ p(x|y) = \frac{p(y|x)p(x)}{p(y)} \propto p(y|x)p(x) \]

- The maximum likelihood approach is applicable to any probability density functions \( p(y|x) \) and \( p(x) \).

- However, let us consider the special case of Gaussian p.d.f’s:

\[
p(x) = \frac{1}{(2\pi)^{N/2}|P^b|^1/2} \exp \left\{ -\frac{1}{2} (x - x_b)^T P^{b-1} (x - x_b) \right\}
\]

\[
p(y|x) = \frac{1}{(2\pi)^{M/2}|R|^1/2} \exp \left\{ -\frac{1}{2} [y - H(x)]^T R^{-1} [y - H(x)] \right\}
\]

- Now, \( J(x) = -\log [p(y|x)] - \log [p(x)] + \text{const} \).

- Hence, with an appropriate choice of the constant \( \text{const} \):

\[
J(x) = \frac{1}{2} (x - x_b)^T P^{b-1} (x - x_b) + \frac{1}{2} [y - H(x)]^T R^{-1} [y - H(x)]
\]

This is the 3D-Var cost function
Maximum Likelihood

Let us introduce the dot product:

$$\langle x_1, x_2 \rangle = x_1^T x_2$$

The dot product is symmetric:

$$\langle x_1, x_2 \rangle = \langle x_2, x_1 \rangle$$

Let us introduce the matrix $A$

$$\langle x_1, Ax_2 \rangle = x_1^T A x_2$$
$$= (A^T x_1)^T x_2$$
$$= \langle A^T x_1, x_2 \rangle$$

$A^T$ is the adjoint of $A$:

$$\langle x_1, Ax_2 \rangle = \langle A^T x_1, x_2 \rangle$$

If $A$ is symmetric ($A^T = A$):

$$\langle x_1, Ax_2 \rangle = \langle Ax_1, x_2 \rangle$$
Maximum Likelihood

The maximum likelihood analysis corresponds to the global minimum of the cost function (using the previously defined dot product):

\[
J(x) = \frac{1}{2} \langle [x - x_b], P^{-1} [x - x_b] \rangle + \frac{1}{2} \langle [y - \mathcal{H}(x)], R^{-1} [y - \mathcal{H}(x)] \rangle
\]

Let introduce a perturbation \( \delta x \) of \( x \). Now, if \( \mathcal{H} \) is linear (or if we neglect second-order terms) then

\[
\mathcal{H}(x + \delta x) = \mathcal{H}(x) + H \delta x.
\]

The cost function evaluated at \( x + \delta x \) is then

\[
J(x + \delta x) = J(x) + \langle \delta x, P^{-1} [x - x_b] \rangle - \langle H \delta x, R^{-1} [y - \mathcal{H}(x)] \rangle
\]

\[
= J(x) + \langle \delta x, P^{-1} [x - x_b] - H^T R^{-1} [y - \mathcal{H}(x)] \rangle
\]

\[
= J(x) + \langle \delta x, \nabla J(x) \rangle.
\]

We deduce the gradient of the cost function

\[
\nabla J(x) = P^{-1} [x - x_b] + H^T R^{-1} [\mathcal{H}(x) - y]
\]
Maximum Likelihood

× At the minimum \( \mathbf{x}_a \), the gradient of the cost function \( \nabla J(\mathbf{x}) \) is zero:

\[
\nabla J(\mathbf{x}_a) = P^{b^{-1}}[\mathbf{x}_a - \mathbf{x}_b] + H^T R^{-1} [\mathcal{H}(\mathbf{x}_a) - \mathbf{y}] = 0
\]

× Now, if \( \mathcal{H} \) is linear (or if we neglect second-order terms) then

\[
\mathcal{H}(\mathbf{x}_a) = \mathcal{H}(\mathbf{x}_b) + H \delta \mathbf{x}_a \quad \text{where} \quad \delta \mathbf{x}_a = \mathbf{x}_a - \mathbf{x}_b
\]

× Hence:

\[
P^{b^{-1}} \delta \mathbf{x}_a + H^T R^{-1} [\mathcal{H}(\mathbf{x}_b) - \mathbf{y}] + H^T R^{-1} H \delta \mathbf{x}_a = 0
\]

× Rearranging a little gives:

\[
\left[ P^{b^{-1}} + H^T R^{-1} H \right] \delta \mathbf{x}_a = H^T R^{-1} [\mathbf{y} - \mathcal{H}(\mathbf{x}_b)]
\]

× This is exactly the equation for the minimum-variance analysis we derived at the start of the lecture!
Maximum Likelihood

We have shown that the maximum likelihood approach is naturally expressed in terms of a cost function representing minus the log of the probability of the analysis state.

The minimum of the cost function corresponds to the maximum likelihood (probability) solution.

For Gaussian errors and linear observation operators, the maximum likelihood analysis coincides with the minimum variance solution.

This is not the case in general:
Maximum Likelihood

- In the nonlinear case, the minimum variance approach is difficult to apply.
- The maximum-likelihood approach is much more generally applicable.
- As long as we know the p.d.f’s, we can define the cost function.
  - However, finding the global minimum may not be easy for highly non-Gaussian p.d.f’s.
- In practice, background errors are usually assumed to be Gaussian (or a nonlinear transformation is applied to make them Gaussian).
  - This makes the background-error term of the cost function quadratic.
- However, non-Gaussian observation errors are taken into account. For example:
  - Directionally-ambiguous wind observations from scatterometers
  - Observations contaminated by occasional gross errors, which make outliers much more likely than implied by a Gaussian model.
Outline

1. From Optimal Interpolation to 3D-Var
2. The Maximum Likelihood Approach
3. Minimisation
4. Summary
Minimisation

- In 3D-Var, the analysis is found by minimising the cost function:

\[ J(x) = \frac{1}{2} (x - x_b)^T P_b^{-1} (x - x_b) + \frac{1}{2} [y - \mathcal{H}(x)]^T R^{-1} [y - \mathcal{H}(x)] \]

- This is a very large-scale \((\text{dim}(x) \approx 10^8)\) minimisation problem.
- The size of the problem restricts on the algorithms we can use.
- Derivative-free algorithms (which require only the ability to calculate \(J(x)\) for arbitrary \(x\)) are too slow.

- This is because each function evaluation gives very limited information about the shape of the cost function.
  - E.g. a finite difference, \(J(x + \delta v) - J(x) \approx \delta v^T \nabla J(x)\), gives a single component of the gradient.
  - We need \(O(10^8)\) components to work out which direction is “downhill”.

©ECMWF
Minimisation

Practical algorithms for minimising the 3D-Var cost function require us to calculate its gradient:

$$\nabla J(x) = P^b -1 (x - x_b) + H^T R^{-1} [H(x) - y]$$

The simplest gradient-based minimisation algorithm is called steepest descent:

Let $x_0$ be an initial guess of the analysis;

while gradient is not sufficiently small do

/* Define a descent direction */
$$d_k = -\nabla J(x_k)$$

/* Find a step $\alpha_k$, e.g. by line minimisation of the function $J(x_k + \alpha d_k)$, for which $J(x_k + \alpha d_k) < J(x_k)$ */
$$\alpha_k = \cdots$$

/* Compute the new estimate */
$$x_{k+1} = x_k + \alpha d_k$$

/* Next step */
$$k = k + 1$$
end
Minimisation

- Steepest descent can work well on problems in which the iso-surfaces of the cost function are nearly spherical.
  - In this case, the steepest descent direction points towards the minimum.
  - They are very well conditioned problems.

- For problems with ellipsoidal iso-surfaces, steepest descent is not efficient.
  - They are poorly conditioned problems.

- We define the curvature as the amount by which a line deviates from being straight.
Preconditioning

- The degree of sphericity of the cost function can be measured by the eigenvalues of the Hessian (matrix $J''$ of second derivatives of $J$).
  - Each eigenvalue corresponds to the curvature in the direction of the corresponding eigenvector.
- The steepest descent method works best if the iso-surfaces of the cost function are approximately spherical.
- This is generally true of all minimisation algorithms.
- In particular, the convergence rate will depend on the condition number:

$$\kappa = \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}}.$$ 

where $\lambda_{\text{max}}$ and $\lambda_{\text{min}}$ are the maximum and minimum eigenvalues respectively.
- In general, expressing the cost function directly in terms of $x$ will not lead to spherical iso-surfaces.
Preconditioning

- We can speed up the convergence of the minimisation by a change of variables $\chi = L^{-1}(x - x_b)$, where $L$ is chosen to make the cost function more spherical.

- A common choice is $L = P^{b1/2}$. The cost function becomes:

$$J(\chi) = \frac{1}{2} \chi^T \chi + \frac{1}{2} [y - H(x_b + L\chi)]^T R^{-1} [y - H(x_b + L\chi)]$$

- With this change of variables, the Hessian becomes:

$$J''_\chi = I + L^T H^T R^{-1} H L \quad \text{(plus higher order terms)}$$

- The presence of the identity matrix in this expression guarantees that the minimum eigenvalue is $\geq 1$.

- There are no small eigenvalues to destroy the conditioning of the problem.
Newton’s methods

- Steepest Descent is inefficient because it does not use information about the curvature of the cost function.
- The simplest algorithms that use curvature are in the family of Newton’s methods.
- Newton’s methods use a local quadratic approximation:

\[
J(x + \delta x) \approx J(x) + \delta x^T \nabla J(x) + \frac{1}{2} \delta x^T J'' \delta x
\]

- Taking the gradient gives:

\[
\nabla J(x + \delta x) \approx \nabla J(x) + J'' \delta x
\]

- Since the gradient is zero at the minimum, Newton’s method chooses the step at each iteration by solving:

\[
J'' \delta x = -\nabla J(x)
\]
Newton’s methods

- Newton’s method:
  Start with an initial guess, $x_0$;
  while gradient is not sufficiently small do
    /* Solve $J''\delta x_k = -\n    \n    \delta x_k = \cdots$;
    /* Compute the new estimate
    $x_{k+1} = x_k + \delta x_k$;
    /* Next step
    $k = k + 1$
  end

- Newton’s method works well for cost functions that are well approximated by a quadratic — i.e. for quasi-linear observation operators.

- However, it suffers from several problems . . .
  - There is no control on the step length $\|\delta x\|$.
  - The method can make huge jumps into regions where the local quadratic approximation is poor.

- This can be controlled using line searches, or by trust region methods that limit the step size to a region where the approximation is valid.
Newton’s methods

Newton’s method requires us to solve $J''\delta x_k = -\nabla J(x_k)$ at every iteration.

Now, $J''$ is a $\sim 10^8 \times 10^8$ matrix! Clearly, we cannot explicitly construct the matrix, or use direct methods to invert it.

However, if we have a code that calculates Hessian-vector products, then we can use an iterative method (e.g. conjugate gradients) to solve for $\delta x_k$.

Such a code is call a **second order adjoint**. See Wang, Navon, LeDimet, Zou, 1992 Meteor. and Atmos. Phys. 50, pp3-20 for details.

Alternatively, we can use a method that constructs an approximation to $(J'')^{-1}$.

Methods based on approximations of $J''$ or $(J'')^{-1}$ are called **quasi-Newton methods**.
Newton’s methods

By far the most popular quasi-Newton method is the BFGS algorithm, named after its creators Broyden, Fletcher, Goldfarb and Shanno.

The BFGS method builds up an approximation to the Hessian:

\[
B_{k+1} = B_k + \frac{y_k y_k^T}{y_k s_k} - \frac{B_k s_k (B_k s_k)^T}{s_k B_k s_k^T}
\]

where \( s_k = x_{k+1} - x_k \) and \( y_k = \nabla J(x_{k+1}) - \nabla J(x_k) \).

The approximation is symmetric and positive definite, and satisfies

\[
\nabla J(x_{j+1}) - \nabla J(x_j) = J''(x_{j+1} - x_j) \quad \text{for} \quad j = 0, 1, \ldots, k
\]

There is an explicit expression for the inverse of \( B_k \), which allows Newton’s equation to be solved at the cost of \( O(Nk) \) operations.
Newton’s methods

The BFGS quasi-Newton method:
Start with an initial guess, \( x_0 \);
Start with an initial approximation of the Hessian (typically, \( B_0 = I \));

while gradient is not sufficiently small do

/* Solve the approximate Newton’s equation, 
\( B_k \delta x_k = -\nabla J(x_k) \), to determine the search direction. */
\( \delta x_k = \cdots; \)

/* Perform a line search to find a step \( \alpha_k \) for which for which 
\( J(x_k + \alpha_k \delta x_k) < J(x_k) \) */
\( \alpha_k = \cdots; \)

/* Compute the new estimate */
\( x_{k+1} = x_k + \alpha_k \delta x_k; \)

/* Generate an updated approximation to the Hessian */
\( B_{k+1} = \cdots; \)

/* Next step */
\( k = k + 1 \)
end
Newton’s methods

The BFGS quasi-Newton method

✖ As $k$ increases, the cost of storing and applying the approximate Hessian increases linearly.

✖ Moreover, the vectors $\mathbf{s}_k$ and $\mathbf{y}_k$ generated many iterations ago no longer provide accurate information about the Hessian.

✖ It is usual to construct $\mathbf{B}_k$ from only the $O(10)$ most recent iterations.

✖ The algorithm is then called the limited memory BFGS method.
Newton’s methods

The methods presented so far apply to general nonlinear functions.

An important special case occurs if the observation operator \( \mathcal{H} \) is linear. In this case, the cost function is strictly quadratic, and the gradient is linear:

\[
\nabla J(x) = P^{-1} b \delta x + H^T R^{-1} [\mathcal{H}(x_b) + H \delta x - y] \\
\]

\[
= \left[ P^{-1} + H^T R^{-1} H \right] \delta x + H^T R^{-1} [\mathcal{H}(x_b) - y] \\
\]

In this case, it makes sense to determine the analysis by solving the linear equation \( \nabla J(x) = 0 \).

Since the matrix \( \left[ P^{-1} + H^T R^{-1} H \right] \) is symmetric and positive definite, the best algorithm to use is conjugate gradients. The algorithm was presented earlier in this lecture.

A good introduction to the method can be found online: Shewchuk (1994) “An Introduction to the Conjugate Gradient Method Without the Agonizing pain”.

©ECMWF
Calculating the Gradient

- To minimise the cost function, we must be able to calculate gradients.
- If we precondition using $L = P^{b^{-1}}$, the gradient (with respect to $\chi$) is:

$$\nabla_{\chi} J(\chi) = \chi + L^T H^T R^{-1} (y - H(x_b + L\chi))$$

- Typically, $R$ is diagonal — observation errors are treated as being mutually uncorrelated.
- However, the matrices $H^T$, $L^T$ and $L$ are not diagonal, and are much too large to be represented explicitly.
- We must represent these as operators (subroutines) that calculate matrix-vector products.
Calculating the Gradient

Take $\mathcal{H}$ as an example. Each line of the subroutine that applies $\mathcal{H}$ can be considered as a function $h_k$, so that

$$\mathcal{H}(x) \equiv h_K(h_{K-1} \cdots (h_1(x)))$$

Each of the functions $h_k$ can be linearised, to give the corresponding linear function $h_k$. Each of these is extremely simple, and can be represented by a one or two lines of code.

The resulting code is called the tangent linear of $\mathcal{H}$.

$$H(x) \equiv h_K h_{K-1} \cdots h_1 x$$

The transpose, $H^T(x) \equiv h_1^T h_2^T \cdots h_K^T x$, is called the adjoint of $\mathcal{H}$.

Again, each $h_k^T$ is extremely simple — just to a few lines of code.

**Tangent Linear and Adjoints**

There is a whole 1-hour lecture on tangent linear and adjoint operators Tuesday when you will learn to derive tangent linear and adjoint equations for a simple nonlinear equation.
Outline

1. From Optimal Interpolation to 3D-Var
2. The Maximum Likelihood Approach
3. Minimisation
4. Summary
Summary

- We showed that 3D-Var can be considered as an iterative procedure for solving the linear (minimum variance) analysis equation.
- We also derived 3D-Var from the maximum likelihood principle.
- The Maximum Likelihood approach can be applied to non-Gaussian, nonlinear analysis.
- We introduced the 3D-Var cost function.
- We considered how to minimise the cost function using algorithms based on knowledge of its gradient.
- We looked at a simple preconditioning.
- Finally, we saw how it is possible to write code that computes the gradient.